UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

ANDERSON VALDIR PERETTO

SISTEMA WEB PARA DIVULGAÇÃO E REPRESENTAÇÃO DE RESTAURANTES

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2017

ANDERSON VALDIR PERETTO

SISTEMA WEB PARA DIVULGAÇÃO E REPRESENTAÇÃO DE RESTAURANTES

Monografia de especialização apresentada na disciplina de Metodologia da Pesquisa, do Curso de Especialização em Tecnologia Java, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientadora: Profa. Andreia Scariot Beulke

PATO BRANCO
2017

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

UTFPR

TERMO DE APROVAÇÃO

DESENVOLVIMENTO DE UM SISTEMA WEB PARA DIVULGAÇÃO E REPRESENTAÇÃO DE RESTAURANTES

por

ANDERSON VALDIR PERETTO

Este trabalho de conclusão de curso foi apresentado em 08 de dezembro de 2017, como requisito parcial para a obtenção do título de Especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Beatriz Terezinha Borsoi e João Guilherme Brasil Pichetti. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

Andreia Scariot Beulke
Profa Orientadora (UTFPR)

Beatriz Terezinha Borsoi
Banca (UTFPR)

João Guilherme Brasil Pichetti
Banca (UTFPR)

Robison Cris Brito
Coordenador da IV Especialização em Tecnologia Java

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

PERETTO, Anderson Valdir. Sistema web para divulgação e representação de restaurantes. 2017. 51f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

Um ambiente virtual proporciona facilidade de acesso aos diversos tipos de informações, sendo necessário, apenas, ter um dispositivo que possua acesso à Internet. Com o advento da Internet é comum a busca por novos produtos, serviços, conteúdos e formas de conhecimento utilizando a *web*. O *marketing* digital tem evoluído com o uso da Internet, pois permite a divulgação de produtos e serviços atingindo uma maior gama de usuários em menos tempo. Considerando esse contexto, o objetivo deste trabalho foi desenvolver um sistema *web* para empresas do segmento de gastronomia, denominadas de restaurantes, e que visa proporcionar a identificação e a localização desses estabelecimentos por seus clientes. O desenvolvimento desse sistema permitiu a integração de diversas tecnologias. Dentre elas destacam-se o *framework* Spring e a linguagem Java para a implementação *back-end*.

Palavras-chave: Restaurantes. Sistema web. *Marketing* digital.

ABSTRACT

PERETTO, Anderson Valdir. System web for disclosure and representation of restaurants. 2017. 51f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

A virtual environment provides easy access to various types of information, only having a device that has access to the Internet. With the advent of the Internet, the search for new products, services, contents and forms of knowledge is common. Digital marketing has evolved with the use of the Internet as it allows the dissemination of products and services reaching a wider range of users in less time. Considering this context, the objective of this work was to develop a web system that presents several companies in the gastronomy segment, called restaurants, and which aims to provide identification and location of these establishments by its customers. The development of this system allowed the integration of several technologies. These technologies include the Spring framework and the Java language for back-end implementation.

Keywords: Restaurants. Web system. Digital marketing.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGOS

LISTA DE SIGLAS

CSS	*Cascading Style Sheet*
DOM	*Document Object Model*
HTML	*Hipertext Markup Language*
IBGE	Instituto Brasileiro de Geografia e Estatística
IDE	*Integrated Development Environment*
JSP	*Java Server Page*
JPA	*Java PersistenceAPI*
JSTL	*JavaServer Pages Standard Tag Library*
POM	*Project Object Model*
RF	Requisito Funcional
RNF	Requisito Não-Funcional
SGBD	Sistema Gerenciador de Banco de Dados
SQL	*Structured Query Language*
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

A facilidade de acesso à informação tem proporcionado aos empreendedores oportunidades para alavancar seus negócios. Nesse aspecto, o *marketing* digital é uma forma de auxiliar os empreendedores por ser econômico promover produtos ou marcas, pois sua execução não demanda grandes orçamentos. Além disso, a facilidade de acesso à Internet proporciona maior gama de visitantes ao site da empresa podendo, assim, convertê-los em clientes. Isso porque as empresas estabelecem um canal de relacionamento direto com os visitantes (por meio de dados cadastrais, se houver), facilitando a comunicação entre o cliente (ou visitante) e a empresa. Assim, o *marketing* digital se tornou um mecanismo importante para divulgação de serviços e para encontrar potenciais clientes no mercado.

De acordo com dados do Instituto Brasileiro de Geografia e Estatística (IBGE), cerca de 95,4 milhões de pessoas tem acesso à Internet no Brasil o que corresponde a aproximadamente 54,4% da população (GOMES, 2016). Dados da pesquisa realizada pela Global News Index indicam que os brasileiros ficam diariamente três horas e quarenta minutos acessando a Internet por meio do celular (FOLHA DE SÃO PAULO, 2015).

Assim, muitos empreendedores resolveram investir em *marketing* digital para divulgar e potencializar suas empresas. Na área gastronômica é comum as empresas investirem em *marketing* digital. Dentre as opções que a tecnologia oferece nessa área, podem ser citados: cardápio digital, promoções, recomendações do *chef*, valor do pedido e tempo de preparo de cada pedido, por exemplo (BEZERRA, LIMA, VIEIRA, 2015). Além disso, pode oferecer a busca por produtos e serviços de melhor qualidade.

De acordo com Bezerra, Lima e Vieira (2015) um restaurante é um local comercial, onde as pessoas procuram por refeições e bebidas. Esses autores, afirmam, ainda que o hábito de alimentar-se fora de casa evoluiu por causa da praticidade e como cultura e lazer. Houaiss e Vilar (2004) enfatizam que um restaurante é um estabelecimento dedicado a servir refeições.

Bezerra, Lima e Vieira (2015) afirmam que dentre os fatores decisivos na escolha de um restaurante destacam-se: as necessidades dos clientes, a qualificação dos cozinheiros, os alimentos oferecidos no cardápio, estratégia de preços, custo do alimento, localização, dentre

outros. Para esses autores, o cardápio é a parte mais importantes para o conceito de restaurante.

Considerando esse contexto, este trabalho apresenta o desenvolvimento de um sistema *web* para representar empresas do segmento de gastronomia, denominadas de restaurantes, e que visa proporcionar a identificação e a localização desses estabelecimentos para seus clientes. Para isso, o sistema permitirá a realização de cadastros, como, por exemplo, de restaurantes que buscam divulgar sua cozinha e cultura gastronômica e de clientes que buscam conhecer esses estabelecimentos. Também permitirá que clientes busquem o restaurante desejado por meio de filtros, avaliar o serviço oferecido e possibilitar a evolução e *marketing* de cozinhas gastronômicas que buscarem crescimento por meio desta plataforma. Além disso, o sistema permitirá que um representante possa optar por representar um restaurante e a quantidade de pratos que cada um deles terá. Para isso, o representante deverá selecionar um plano que terá valores diferenciados conforme a opção escolhida.

2 FERRAMENTAS, TECNOLOGIAS E PROCEDIMENTOS

Este capítulo apresenta os materiais e o método utilizados para o desenvolvimento deste trabalho. Os materiais elencam as ferramentas e as tecnologias utilizadas para modelagem, banco de dados, linguagem de programação e *frameworks* para programação *front* e *back-end*. O método está relacionado a um modelo de processo prescritivo que visa ordenar e estruturar o desenvolvimento do aplicativo.

2.1 FERRAMENTAS E TECNOLOGIAS

O Quadro 1 apresenta as ferramentas e tecnologias utilizadas na modelagem e no desenvolvimento do aplicativo proposto neste trabalho.

Ferramenta / Tecnologia	**Versão**	**Disponível em**	**Aplicação**
MySQL Server	5.7	https://dev.mysql.com/downloads/mysql/	Sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem *Structured Query Language* (SQL).
MySQL Workbench	6.3	https://dev.mysql.com/downloads/mysql/	Modelagem do banco de dados do sistema.
Java EE	8.0	http://www.oracle.com/technetwork/java/javaee/downloads/index.html	Linguagem para desenvolvimento da aplicação.
HTML	5.0	https://www.w3.org/TR/html51/	Linguagem de marcação de textos utilizada para desenvolvimento de interfaces de aplicações.
CSS	3	https://www.w3.org/Style/CSS/	Estilização das páginas.
Bootstrap	3.3.6	http://getbootstrap.com/	*FrameworkHipertext Markup Language* (HTML), *Cascading Style Sheet* (CSS) e JavaScript.
Hibernate	5.1.0	http://Hibernate .org/	Para mapeamento objeto relacional e persistência de dados.
JQuery	2.2.4	https://jquery.com/	Biblioteca JavaScript utilizada no desenvolvimento da interface.
Eclipse	Neon	https://eclipse.org/	*Integrated Development Environment*

			(IDE) para desenvolvimento da aplicação.
Spring Framework	5.0.2	http://spring.io/	*Framework* para desenvolvimento ágil de sistemas *web* com a linguagem de programação Java.
Apache TomCat	8.5	http://tomcat.apache.org/	Container de *servlets* que implementa as tecnologias Java, funciona como um servidor para aplicações em Java.
Maven	4.0	https://maven.apache.org/	Modelagem do projeto e gerenciamento de dependências.
FontAwesome	4.6.3	http://fontawesome.io/	Fonte de ícones vetorizados.
JQuery Mask *Plugin*	1.7.6	https://plugins.jquery.com/mask/	*Plugin* para JQuery para atribuição de máscaras nos campos de um formulário.

Quadro 1 - Tecnologias e ferramentas utilizadas na modelagem e na implementação do aplicativo

Para configuração e gerenciamento das dependências do projeto foi utilizado o Maven 4.0. Toda a configuração é realizada por meio do arquivo *Project Object Model* (POM). Esse arquivo está presente em todo o projeto, pois ele contém a identificação, as dependências e repositório adicional, por exemplo.

Foram utilizados, ainda, alguns *frameworks* sendo o principal deles o Spring na versão 5.0.2. O Spring fornece um modelo abrangente de programação e configuração para aplicativos corporativos baseados em Java para qualquer plataforma de implantação. Um elemento-chave do Spring é o suporte infraestrutural no nível de aplicação: o Spring se concentra no "encanamento" de aplicativos para que as equipes possam se concentrar na lógica comercial de nível de aplicativo, sem vínculos desnecessários com ambientes de implantação específicos.

O *framework* Hibernate 4.3.1 foi utilizado para o mapeamento objeto relacional. A principal função desse *framework* é abstrair o mapeamento objeto-relacional por meio de *annotations*. Assim, as classes em Java são transformadas para tabelas de dados, gerando as chamadas SQL.

O *framework* Boostrap e a biblioteca JQuery foram utilizados para otimizar o desenvolvimento *front-end*. O Bootstrap visa auxiliar no desenvolvimento de páginas *web* responsivas fornecendo recursos HTML, CSS e JavaScript. Ele inclui classes predefinidas

para opções de leiaute. Dentre os seus recursos, destaca-se o sistema de *grids* que utiliza linhas e colunas para definição do leiaute. O JQuery utiliza seletores que permitem recuperar, com mais facilidade, elementos do *Document Object Model* (DOM), além de permitir a seleção e a manipulação de elementos HTML e CSS, possuir efeitos de animação e eventos.

2.2 PROCEDIMENTOS TÉCNICOS

Para desenvolver o trabalho com as ferramentas apresentadas na Seção 2.1, deve-se instalar o Eclipse Neon, distribuição Java EE incluindo o Apache TomCat 8.0.27. Para criar um projeto novo que faça uso dos recursos do Spring são necessárias algumas configurações adicionais:

1) Criar um novo projeto Maven de aplicação *web* e estruturar os diretórios conforme apresentado na Figura 1.

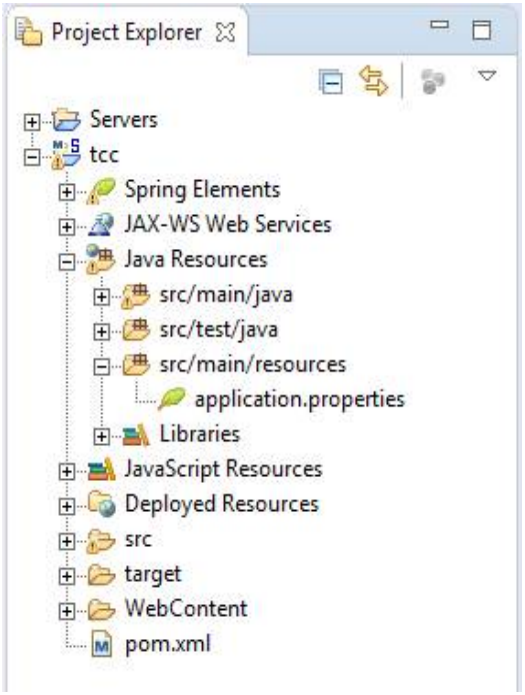

Figura 1 - Estrutura do projeto

2) Criar o arquivo “application.properties” e inserir o código apresentado na Listagem 1 para que seja criada a comunicação com o banco de dados. Caso o usuário e a senha configurados sejam outros, deve-se alterar.

```
# JDBC PROPERTIES
jdbc.user = root
jdbc.pass = root
jdbc.url = jdbc:mysql://localhost:3306/tcc
jdbc.driverClass = com.mysql.jdbc.Driver

# HIBERNATE PROPERTIES
hibernate.show.sql = true
hibernate.ddl = true
hibernate.package.scan = br.edu.utfpr.tcc.entity
```

Listagem 1- Arquivo Applications.properties

3) Acessar o MySql Workbench e criar um “schema” com o nome do projeto que será a base de dados.

4) Acessar o arquivo pom.xml e adicionar as dependências necessárias para o desenvolvimento do projeto.

3 RESULTADOS

Este capítulo apresenta o resultado da realização do trabalho que é o desenvolvimento de um sistema *web* para divulgação e representação de restaurantes.

3.1 ESCOPO DO SISTEMA

O sistema *web*, desenvolvido como resultado deste trabalho, visa oferecer mais comodidade aos usuários que têm como objetivo encontrar locais para realizar refeições, denominados genericamente, neste texto, de restaurantes. São locais que oferecem refeições diárias, almoços especiais de final de semana e gastronomias mais sofisticadas. O sistema atuará como *marketing* para divulgação de restaurantes, enfatizando facilidade de uso e a localização desses ambientes pelos usuários interessados.

O sistema permite que seja realizado o cadastro do representante que será a pessoa encarregada da manutenção e da apresentação do restaurante, o cadastro do restaurante, informando dados, como, por exemplo, endereço completo e a área gastronômica de atuação do estabelecimento. Também será possível escolher planos (previamente cadastrados pelo responsável pelo sistema) que conterá a quantidade de pratos que um representante poderá cadastrar para o referido restaurante e também o valor do plano. Após ser escolhido o plano, poderá ser realizado o cadastro das variações de pratos oferecidos pelo restaurante. A busca por um restaurante será realizada com base no cadastro de tipos de pratos que conterá a descrição e informações consideradas essenciais para uma busca mais efetiva pelo cliente.

O cliente poderá realizar busca mesmo que não tenha cadastro. Contudo, com acesso restrito de informações. O cliente poderá fazer sua inscrição sem custo, informando seus dados pessoais, endereço completo e a área gastronômica que deseja pesquisar.

O sistema permite que sejam realizadas buscas por restaurantes. Essas buscas são baseadas em filtros, podendo buscar pela localidade. Caso o usuário esteja em um lugar que não conhece muito bem, poderá também fazer a busca informando sua localização e o sistema apresentará os restaurantes mais próximos ao usuário. Também será possível fazer busca pelo

nome do restaurante. O sistema também permite que os clientes possam realizar comentários sobre o restaurante.

3.2 MODELAGEM DO SISTEMA

O processo de desenvolvimento do sistema foi realizado com base no modelo da Engenharia de Software denominado iterativo e incremental, no qual cada funcionalidade é dividida em ciclos chamados de incrementos e cada ciclo possui seis etapas que são: levantamento de requisitos, análise dos requisitos, projeto/modelagem, codificação, testes e implantação (SOMMERVILLE, 2003).

O Quadro 2 apresenta os requisitos funcionais do sistema. Nesse quadro, a sigla RF se refere a Requisito Funcional.

Identificação	Nome	Descrição
RF 01	Manter usuário	Manter informações dos usuários do sistema, contendo os dados necessários para identificação, *login* e a definição das prioridades de acesso. Os usuários do sistema são: administrador, representante e cliente.
RF 02	Manter planos	O plano se refere à quantidade de pratos que poderão ser cadastrados à um restaurante e o valor que o seu representante terá como despesa de mensalidade.
RF 03	Manter restaurante	Manter informações dos restaurantes que poderão ser pesquisados e avaliados pelos clientes.
RF 04	Contratar plano	O representante poderá contratar um plano para apresentar o restaurante.
RF 05	Manter categoria de prato	Refere-se aos tipos de pratos oferecidos pelo restaurante, como, por exemplo, massas, carnes, entre outros.
RF 06	Manter prato	Cadastro de pratos que serão oferecidos pelos restaurantes.
RF 07	Realizar comentários sobre os restaurantes	Os usuários poderão fazer comentários sobre os restaurantes.

Quadro 2- Requisitos Funcionais

O Quadro 3 e apresenta os requisitos não funcionais do sistema que são representados pela sigla RNF.

Identificação	Nome	Descrição
RNF 01	Acesso ao sistema	O acesso ao sistema será realizado por meio de *login* e senha. Podendo também realizar acesso sem login, sendo então um acesso anônimo, este com acesso limitado.
RNF 02	Restrições de formato	Os campos de *e-mail* e senha devem ser válidos de acordo com as restrições de formato.
RNF 03	Campos de preenchimento obrigatórios	Os campos que são de preenchimento obrigatório serão validados por meio de uma função do sistema.
RNF 04	Campos com máscaras de entrada	Os campos que possuem caracteres especiais e, que não serão armazenados no banco de dados, como, por exemplo, "()", ".", "-", para os campos CPF, CNPJ, telefone, entre outros, serão validados por meio de máscaras de entrada.
RNF 05	Validação (manter pratos)	Os pratos só podem ser cadastrados se já houver algum plano contratado pelo representante.

Quadro 3 - Requisitos Não Funcionais

A Figura 2 apresenta o diagrama de casos de uso que contém as funcionalidades essenciais do sistema realizadas pelos seus atores que são: administrador, representante e cliente. O administrador é responsável pelos cadastros de categorias de pratos e planos que serão contratados pelos representantes. O representante poderá cadastrar os dados do restaurante e seus respectivos pratos (de acordo com o plano contratado). O cliente é responsável por pesquisar e realizar comentários sobre um restaurante. A pesquisa por restaurantes ocorrerá por meio de filtros (filtros de busca pela descrição e localização). A avaliação ocorrerá por meio de comentário.

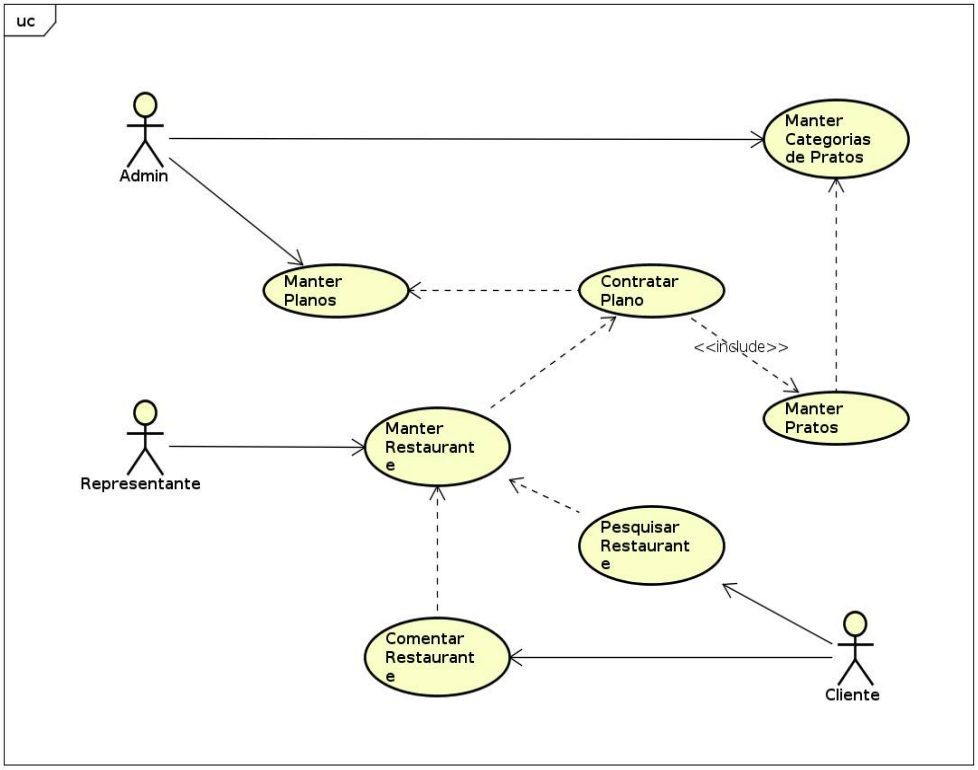

Figura 2 - Diagrama de Caso de Uso

Os quadros numerados de 4 a 11 apresentam a expansão dos casos de uso. Os casos de uso dos quadros 4 a 7 apresentam a descrição dos casos de uso do tipo “Manter” que incluem as operações de inserir, consultar, alterar e excluir dados de um objeto do sistema. Essas operações apresentam, basicamente, o mesmo comportamento para todos os casos de uso que incluem essas atividades.

Caso de uso:
Incluir (refere-se à operação de inclusão nos casos de uso “manter”).
Descrição:
Ator inclui dados no sistema.
Atores:
Administrador ou representante de acordo com suas funções definidas no caso de uso.
Pré-condição:
Estar autenticado no sistema.
Sequência de Eventos:
1. Ator acessa a página do sistemapara realizar o cadastro.

2. Ator preenche os campos e clica no botão “Salvar”.
3. O sistema insere as informações no banco de dados e mostra mensagem de confirmação da operação.

Pós-Condição:

Registro inserido no banco de dados.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1. O ator não preenche os campos obrigatórios e clica no botão “Salvar”. 1.2 O sistema valida as informações e exibe mensagem informando que os campos obrigatórios não foram preenchidos corretamente. 1.3. O sistema retorna à tela de inclusão, com os campos que haviam sido preenchidos e destacando os campos obrigatórios sem preenchimento.
2. Campos preenchidos com formato inválido	2.1. O ator preenche os campos de forma incorreta e clica no botão “Salvar”. 2.2. O sistema valida as informações e exibe uma mensagem informando que os dados foram preenchidos de forma incorreta. 2.3. O sistema retorna à tela de inclusão, com os campos que já haviam sido preenchidos e destacando os campos com formato inválido.

Quadro 4- Operação incluir dos casos de uso Manter

Caso de uso:

Alterar (refere-se à operação de alteração nos casos de uso “manter”).

Descrição:

Ator altera dados no sistema.

Atores:

Administrador ou representante de acordo com suas funções definidas no caso de uso.

Pré-condição:

Dados cadastrados no sistema.

Sequência de Eventos:

1. O ator acessa a tela para visualização de dados já cadastrados.
2. O sistema apresenta o registro selecionado para a alteração.
3. Ator altera os dados do registro e clica em salvar.
4. O sistema valida as informações e salva no mesmo registro.

Pós-Condição:

Registro alterado no banco de dados.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1. O ator não preenche campos obrigatórios e clica no botão “Salvar”. 1.2. O sistema valida as informações e exibe uma mensagem

	informando que campos obrigatórios não foram preenchidos e não salva as alterações no banco de dados. 1.3. O sistema retorna a tela de alteração, com os campos que já haviam sido preenchidos e destacando os campos sem preenchimento.
2. Campos preenchidos com formato inválido.	2.1. O ator preenche os campos de forma incorreta e clica no botão "Salvar". 2.2. O sistema exibe mensagem informando que os dados não estão no formato e não salva o registro. 2.3. O sistema retorna à tela de alteração, com os campos que já haviam sido preenchidos corretamente e destaca os campos com formato inválido.

Quadro 5- Operação alterar dos casos de uso Manter

Caso de uso:
Excluir (refere-se à operação de exclusão nos casos de uso "manter").
Descrição:
Ator solicita a exclusão de dados no sistema.
Atores:
Administrador ou representante de acordo com suas funções definidas no caso de uso.
Pré-condição:
Dados cadastrados no sistema.
Sequência de Eventos:
1. O ator acessa a tela para visualização de dados já cadastrados.
2. O sistema apresenta o registro selecionado para a exclusão.
3. Ator clica em excluir registro.
4. O sistema exclui as informações do banco de dados e exibe as informações do *status* do procedimento.

Pós-Condição:
Registro excluído do banco de dados.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Exclusão de registro que possui vínculos no sistema.	1.1. Ator solicita a exclusão do registro que possui vínculos no sistema. 1.2 O sistema verifica se o registro possui vínculos e, em caso positivo exibe uma mensagem informando que o registro não pode ser excluído.

Quadro 6- Operação excluir dos casos de uso Manter

Caso de uso:
Consultar (refere-se à operação de consulta nos casos de uso "manter").
Descrição:
Ator solicita a consulta de dados cadastrados no sistema.
Atores:

Cliente, administrador ou representante de acordo com suas funções definidas no caso de uso.

Pré-condição:

Dados cadastrados no sistema.

Sequência de Eventos:

1. Ator acessa a tela para visualização de dados já cadastrados.
2. Ator indica quais dados pretende consultar por meio de filtros.
3. O sistema exibe os dados da consulta ao usuário.

Pós-Condição:

Dados são exibidos aos usuários.

Quadro 7- Operação consultar dos casos de uso Manter

O Quadro 8 apresenta a especificação do caso de uso denominado contratar plano.

Caso de uso:

Contratar plano

Descrição:

Ao cadastrar um restaurante, o representante deve selecionar um plano e o contratar.

Atores:

Representante.

Pré-condição:

Representante deve estar autenticado no sistema e com o Restaurante cadastrado.

Sequência de Eventos:

1. Representante acessa a página com a lista de planos.
2. Representante clica na opção "+ Detalhes" no plano desejado.
3. Sistema exibe tela com os detalhes do plano selecionado.
4. Representante clica no botão Contratar.
5. Sistema encaminha para a página de perfil do representante com o plano já contratado.

Pós-Condição:

Contrato do plano efetuado.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Planos não disponíveis	1.1 Sistema não irá apresentar o botão "Contratar" no perfil do plano se este não estiver disponível para contratação.

Quadro 8- Caso de uso Contratar Plano de Quantidade de Pratos

O Quadro 9 apresenta o caso de uso denominado de pesquisar restaurante.

Caso de uso:

Pesquisar Restaurante

Descrição:

Usuário pode realizar pesquisas sobre restaurantes. Para realizar pesquisa sobre restaurante e verificar seus detalhes qualquer usuário pode fazer este processo, inclusive usuário anônimo (usuário não cadastrado no sistema), mas para obter detalhes sobre pratos e

comentar sobre os restaurantes necessita-se estar cadastrado no sistema e acessar com seu devido login.

Atores:

Cliente.

Pré-condição:

Devem haver restaurantes já cadastrados.

Sequência de Eventos:

1. Cliente acessa o sistema.
2. Cliente informa sua pesquisa no campo de busca.
3. Cliente utiliza os filtros para encontrar o restaurante desejado.

Pós-Condição:

Sistema exibe lista com os dados encontrados.

Quadro 9- Caso de uso Pesquisar Restaurantes

O Quadro 10 apresenta a especificação do caso de uso denominado de comentar restaurante.

Caso de uso:

Comentar Restaurante

Descrição:

Cliente pode comentar o restaurante desejado ao acessar o sistema.

Atores:

Cliente.

Pré-condição:

Cliente precisa estar autenticado no sistema para poder avaliar o restaurante.

Sequência de Eventos:

1. Cliente acessa a página do restaurante.
2. Cliente pode informar seu comentário sobre o restaurante.

Pós-Condição:

Avaliação de restaurante concluída.

Quadro 10- Caso de uso Avaliar Restaurante

A Figura 3 apresenta o diagrama de classes do sistema. As entidades e relacionamentos funcionam da seguinte forma:

a) a entidade denominada restaurante é uma representação dos dados comuns de todos os restaurantes.

b) a entidade denominada pratos representa um item que está diretamente associada a um restaurante e, por isso, possui alguns atributos exclusivos, como, por exemplo, os atributos obrigatórios nome, descrição, valor, sobre, ingredientes, avatar e o próprio restaurante.

c) a entidade denominada de avatares é uma tabela comum para várias outras entidades, a qual é utilizada para armazenar as imagens de usuários, de pratos e de restaurantes.

d) a entidade de categorias possui relacionamento direto com pratos, pois é de uso obrigatório no momento de cadastrar um novo prato no sistema.

e) a entidade de comentários refere-se à de restaurante, na qual os usuários cadastrados podem realizar comentários sobre o restaurante desejado.

f) a entidade de usuário é a responsável pela manutenção do sistema, pois os usuários poderão alimentá-lo com dados e interagir com o mesmo. Os usuários são caracterizados como administrador, representante e cliente.

g) a entidade de representantes é responsável pela criação e manutenção de restaurantes.

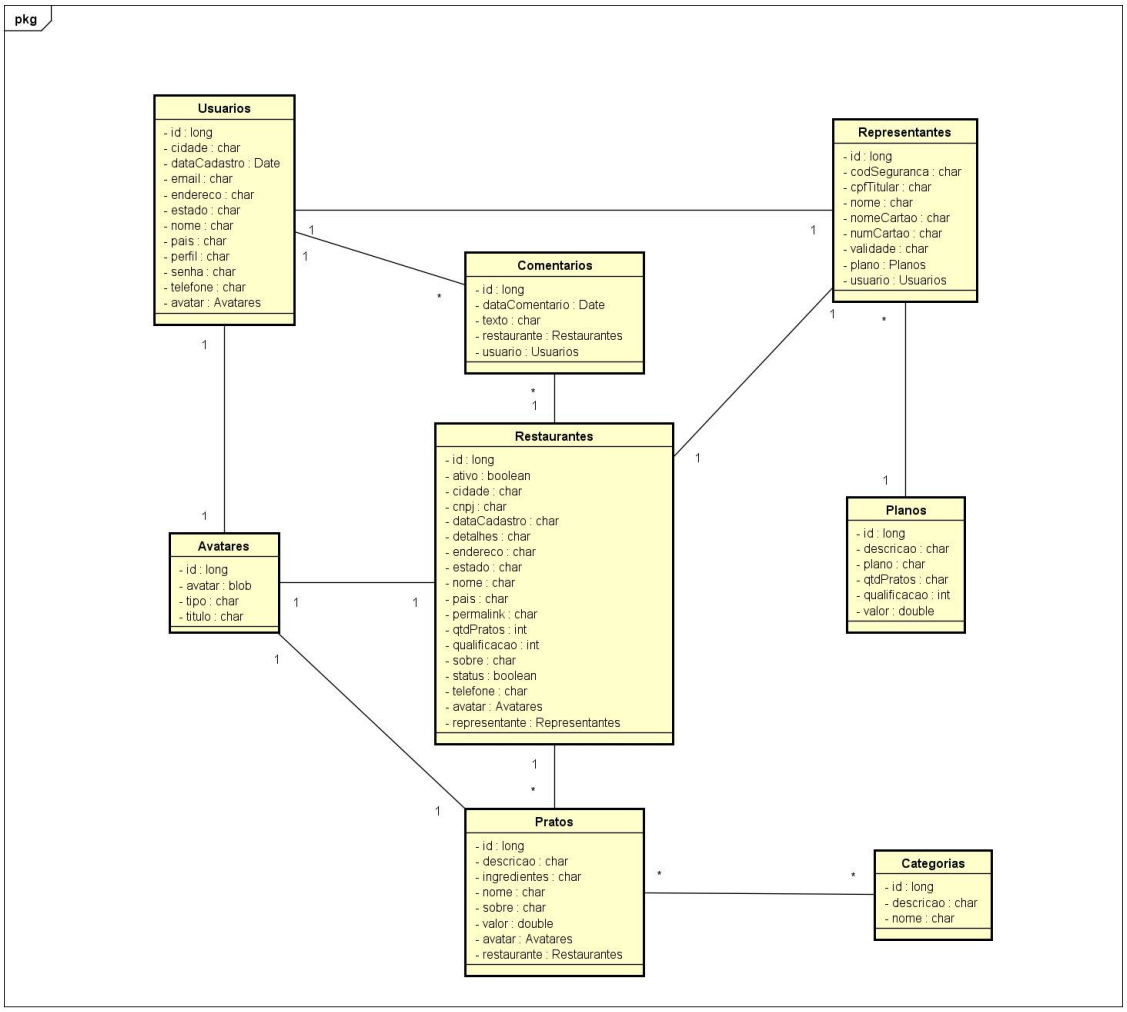

Figura 3 - Diagrama de Classe

A Figura 4 apresenta o diagrama de entidade e relacionamento do banco de dados e segue o mesmo padrão de relacionamentos do diagrama de classes.

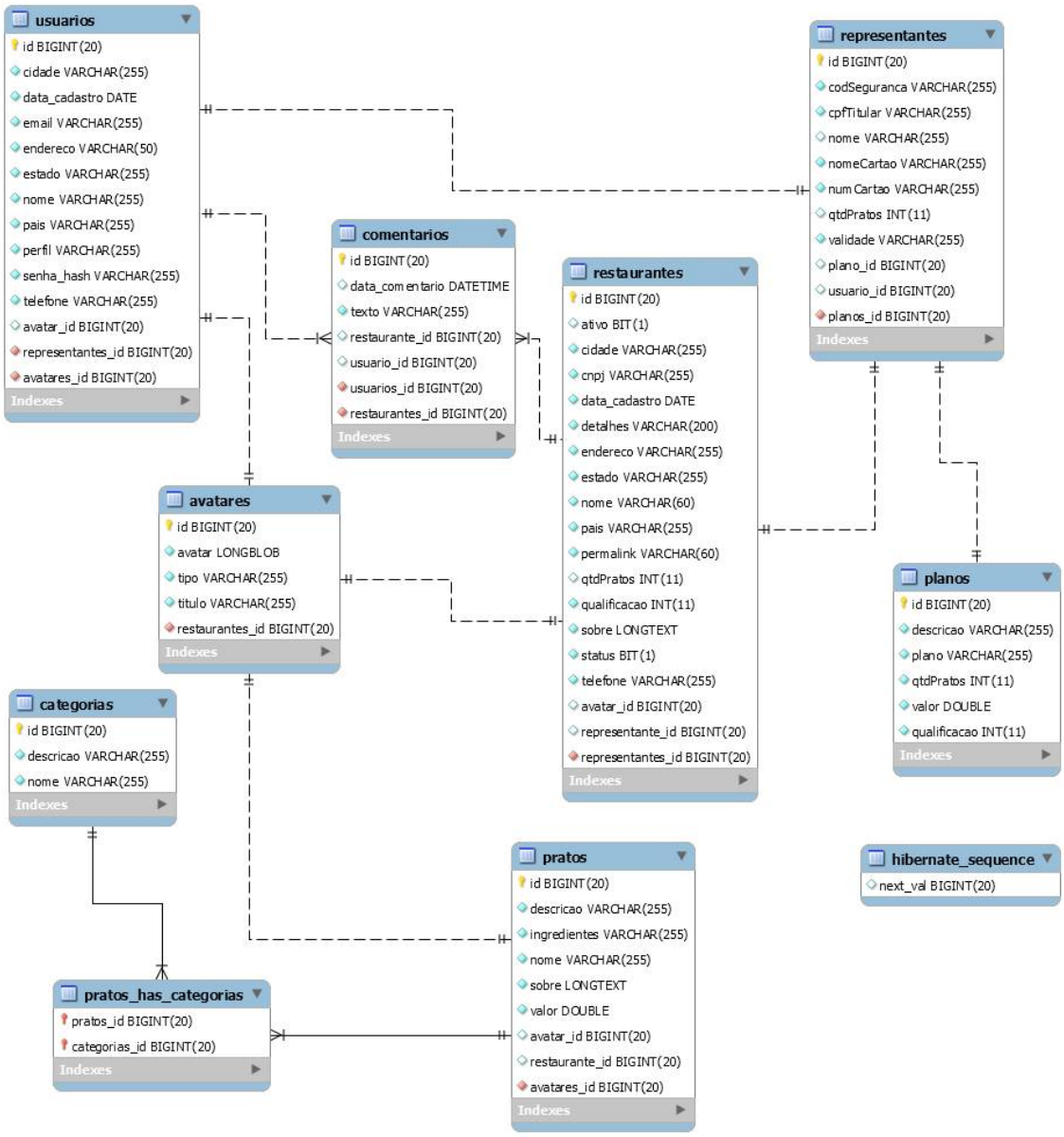

Figura 4 - Diagrama de Entidade e Relacionamento do Banco de Dados

3.3 APRESENTAÇÃO DO SISTEMA

A Figura 5 apresenta a tela de login do sistema cujo acesso é realizado por meio das informações de e-mail e senha. Caso o usuário não possua uma conta será necessário criá-la por meio do link “Crie uma agora” apresentado na Figura 5.

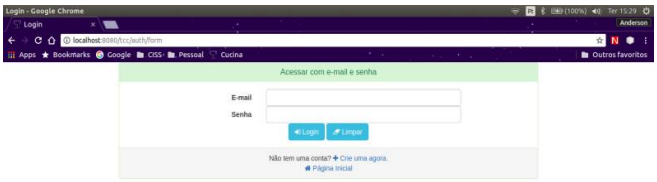

Figura 5 - Tela de autenticação

A Figura 6 apresenta a tela de busca de restaurantes, essa busca pode ser feita pela tela inicial do sistema ou também pela tela de lista de Restaurantes. A busca é feita por palavras chaves, podendo ser pelo nome do restaurante ou por cidade.

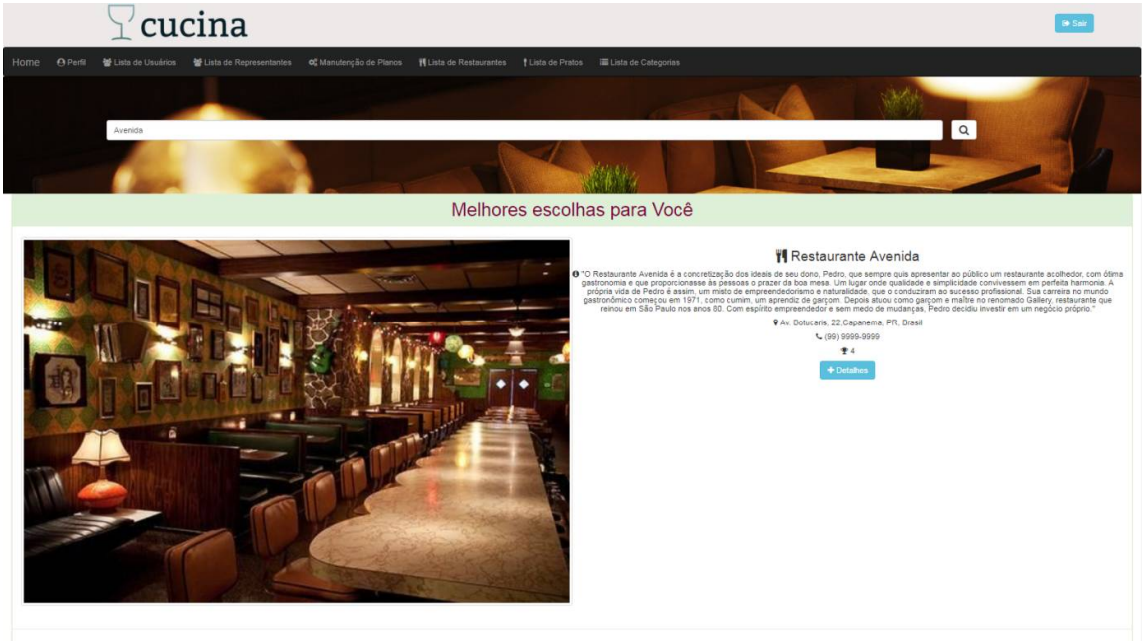

Figura 6 - Tela de busca de Restaurante

A Figura 7 apresenta a tela de cadastro de usuário. Nesta tela o usuário pode efetuar seu cadastro no sistema informando seus dados pessoais, cadastrando-se com perfil de cliente. Para se cadastrar como representante o usuário deve clicar no link “Cadastrar-se como representante” no rodapé da página que será direcionado para a tela de cadastro de usuário com perfil de representante. O cadastro de usuários com perfil de administrador só pode ser feito por um usuário administrador já cadastrado através da lista de usuários. O perfil de cliente permite o acesso às informações detalhadas sobre restaurantes e seus pratos e a realização de comentários sobre os restaurantes. O perfil de representante permite que seja cadastrado um restaurante que o representante representar por meio do contrato e manutenção de planos de uso e cadastro de pratos para seu respectivo restaurante.

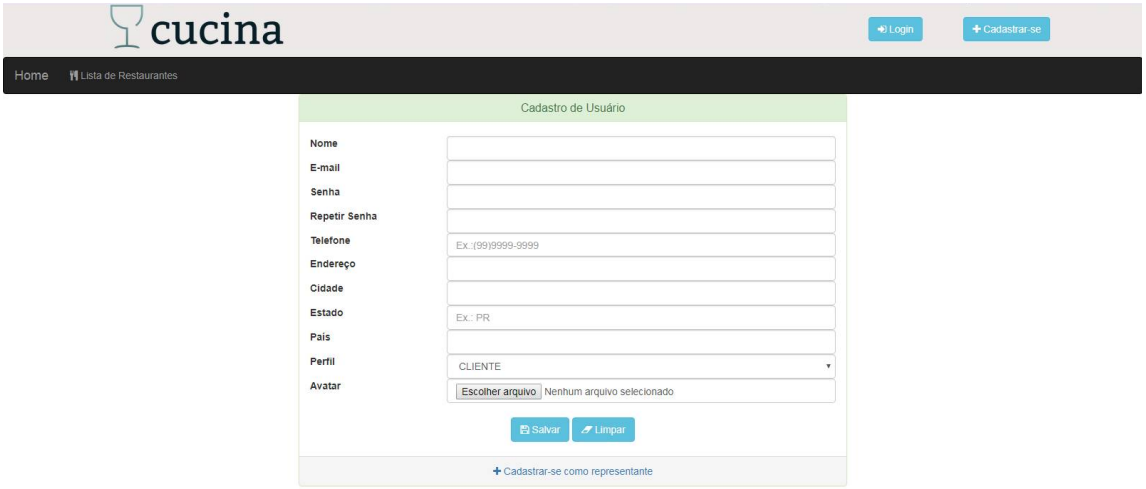

Figura 7 - Tela de cadastro de usuário

A Figura 8 apresenta a tela de perfil de usuário. Nesta tela o usuário pode verificar suas informações cadastrais, editar e excluir o seu perfil.

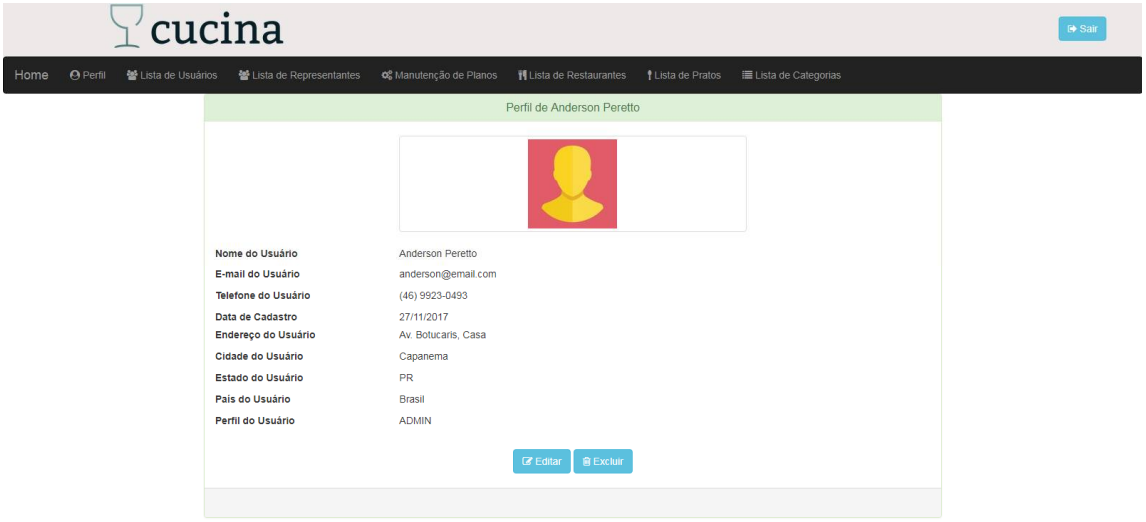

Figura 8 - Tela de perfil de usuário

A Figura 9 apresenta a tela de edição do perfil de usuário. Nesta tela o usuário pode editar suas informações cadastrais. Na tela existem três quadros de edição: avatar, senha e cadastro de dados pessoais do usuário.

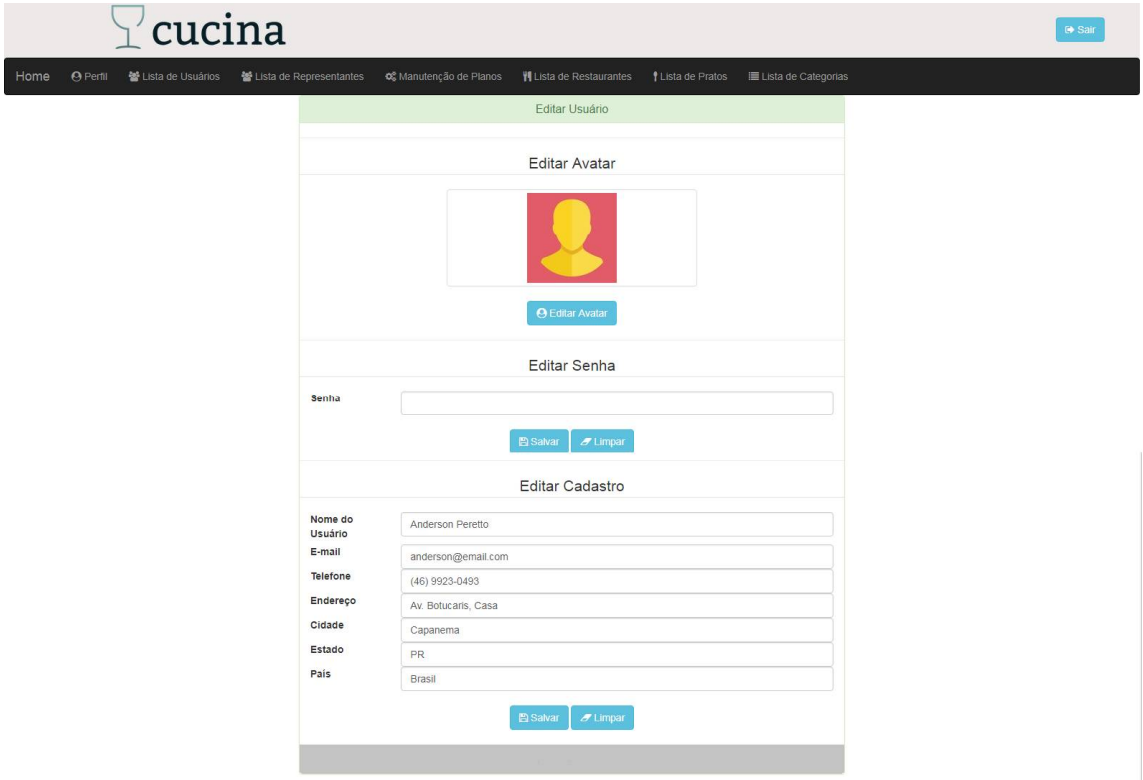

Figura 9 – Tela de edição de usuário

A Figura 10 apresenta a lista de usuários em ordem alfabética de nome. Apresenta, também, a opção de busca por nome de usuário. Essa tela é disponível apenas para usuários administradores.

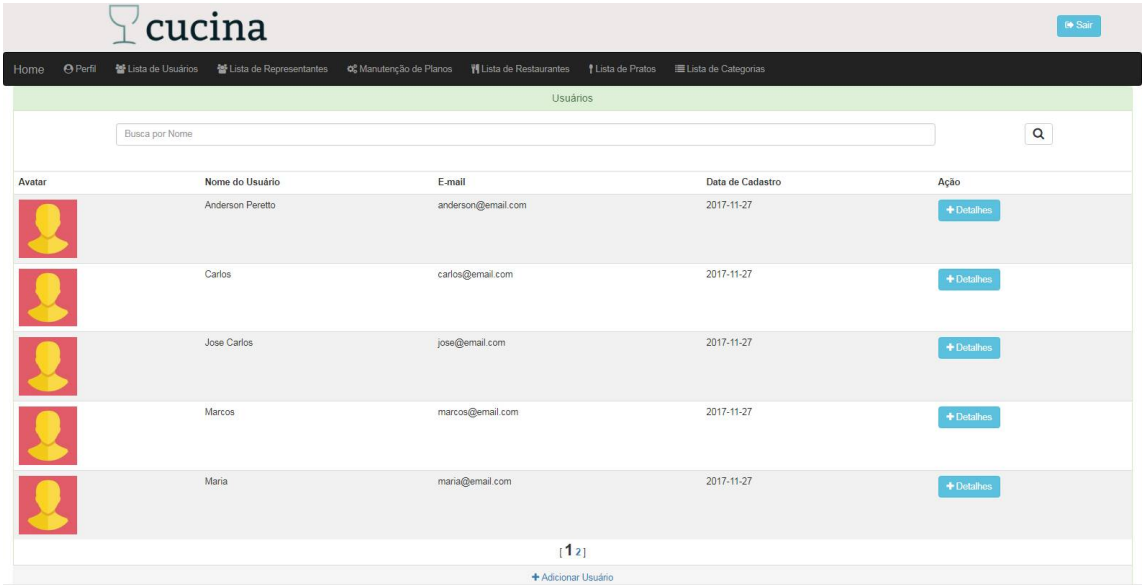

Figura 10 – Tela de listagem de usuários

A Figura 11 apresenta a tela inicial do sistema para o usuário com perfil de representante. Na parte superior encontra-se o menu com as permissões de acesso, seu perfil de usuário ou representante e os dados do restaurante do qual ele é representante, manutenção de planos e também as opções de buscas de restaurantes.

Figura 11 - Tela Inicial do Sistema

A Figura 12 apresenta a tela de cadastro de representante que deve informar seu nome e os dados do cartão de crédito para faturar o plano a ser contratado.

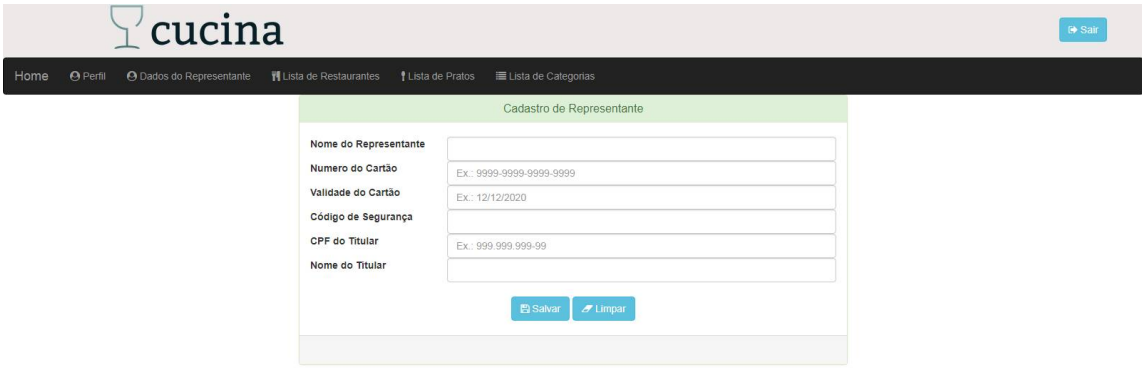

Figura 12 – Tela de cadastro de representante

A Figura 13 apresenta a tela de perfil de representante. Nessa tela o representante pode verificar suas informações, editá-las ou excluir seu registro.

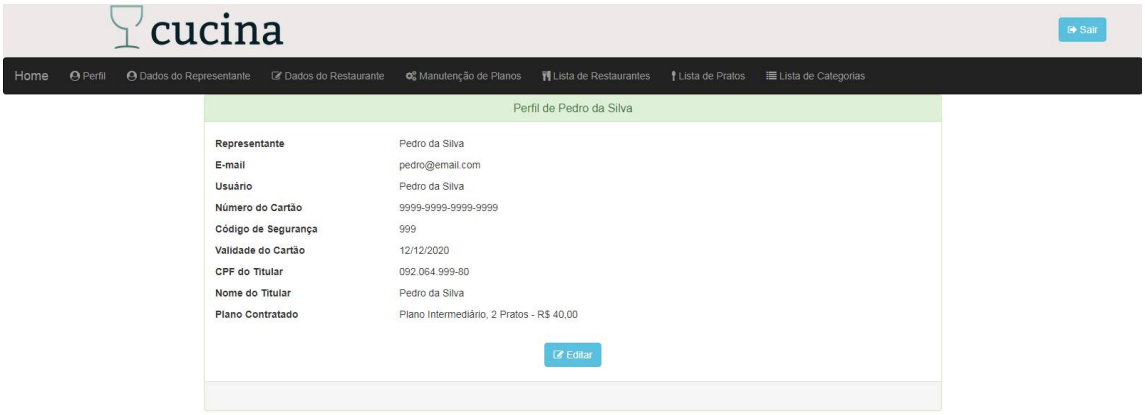

Figura 13 – Tela de perfil de representante

Ao clicar no botão editar o sistema exibe a tela com as informações preenchidas, permitindo que o representante possa modificá-las por meio da alteração dos campos e salvá-las, clicando no botão salvar, conforme exposto na Figura 14.

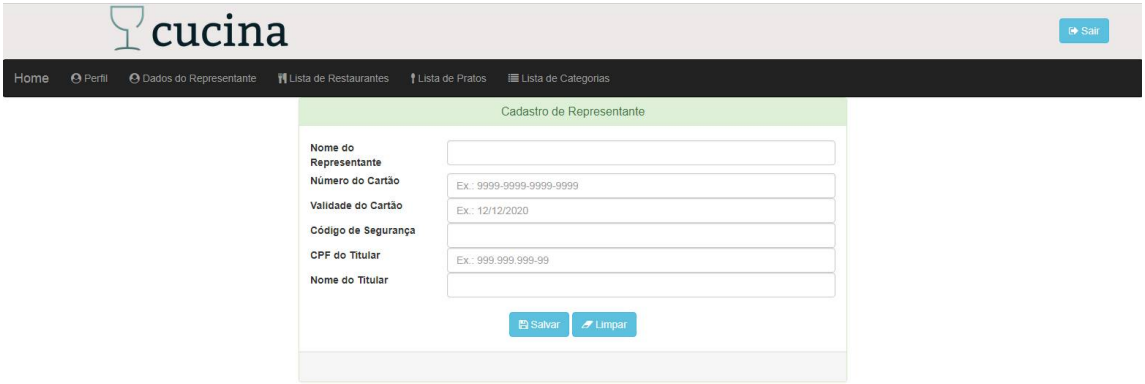

Figura 14 – Tela de edição de representante

A Figura 15 apresenta a lista de representante, por ordem alfabética de nome e a opção de busca por nome de usuário.

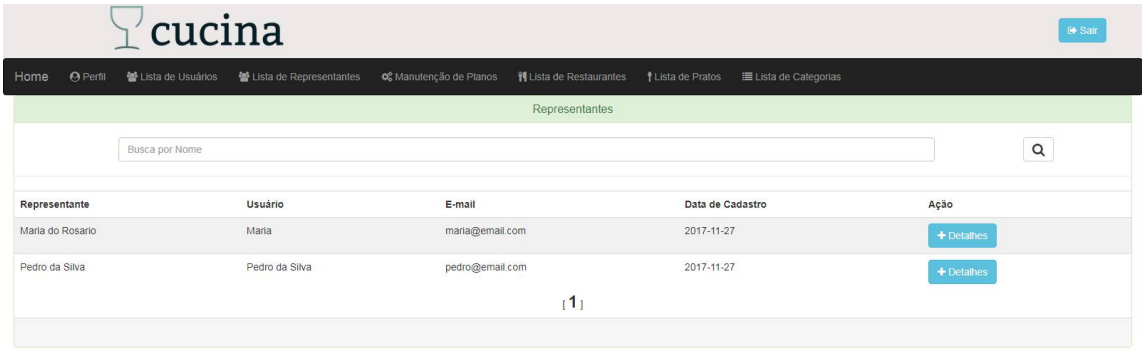

Figura 15 – Tela de listagem de representante

A Figura 16 apresenta a tela de detalhes do restaurante que possui a listagem de pratos e a de comentários realizados por usuários do restaurante.

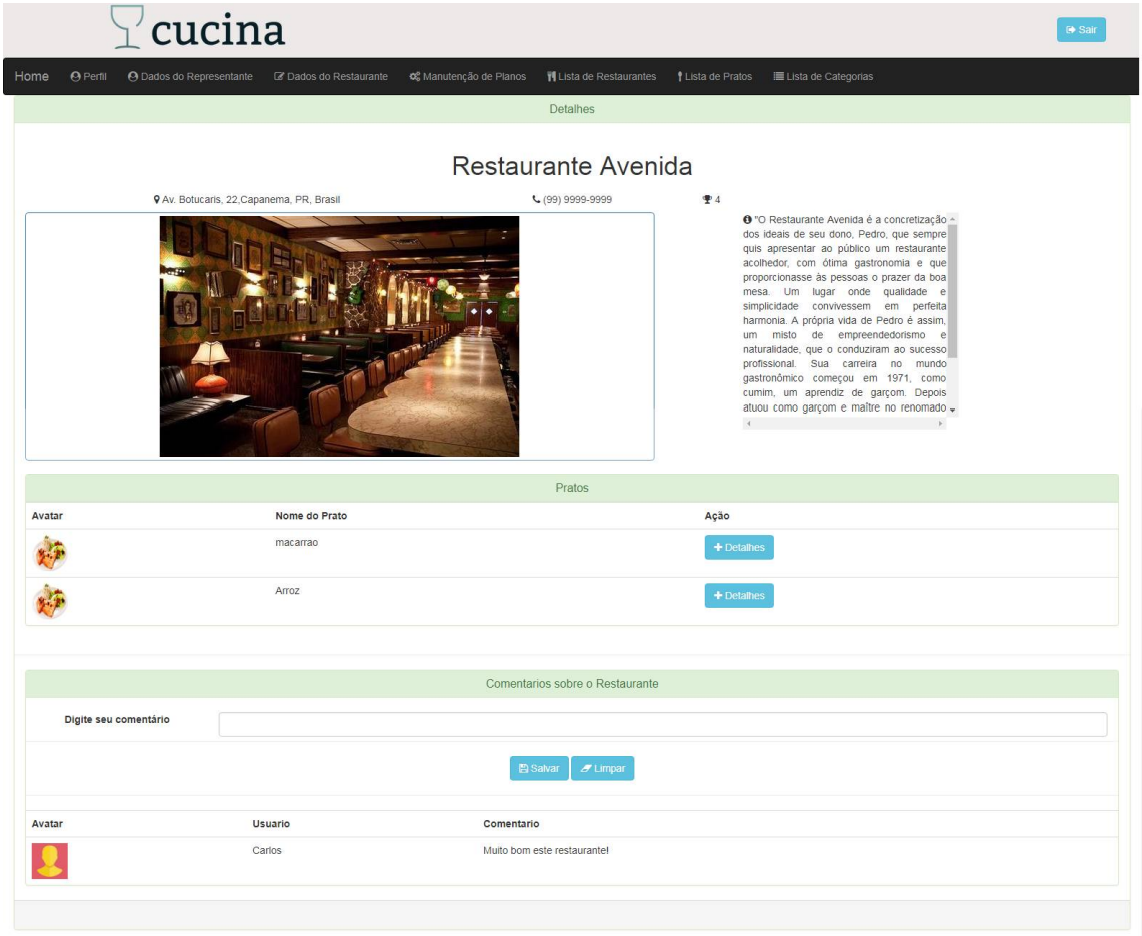

Figura 16 – Tela de detalhes do restaurante

Após o representante ter realizado o contrato de um plano, ele poderá cadastrar os pratos do restaurante que representa. Para isso, basta informar os dados do prato, como, nome, descrição, ingredientes, informações sobre o prato, valor, categoria à qual o prato pertence e escolher uma imagem para ilustrar o prato, conforme apresentado na Figura 17.

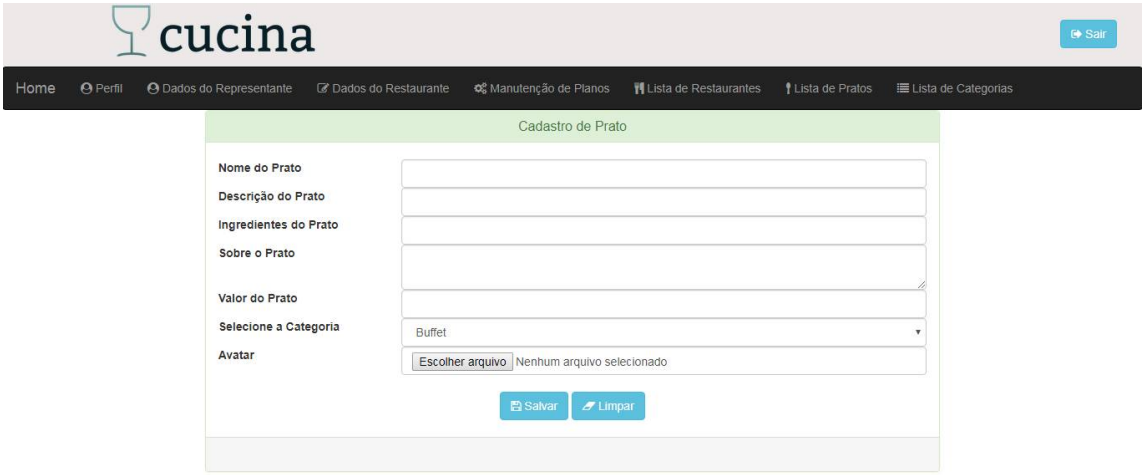

Figura 17 – Tela de cadastro de pratos

Após cadastrar o prato, é possível visualizar as informações por meio de um grid com todos os pratos registrados no banco de dados, conforme apresentado na Figura 18.

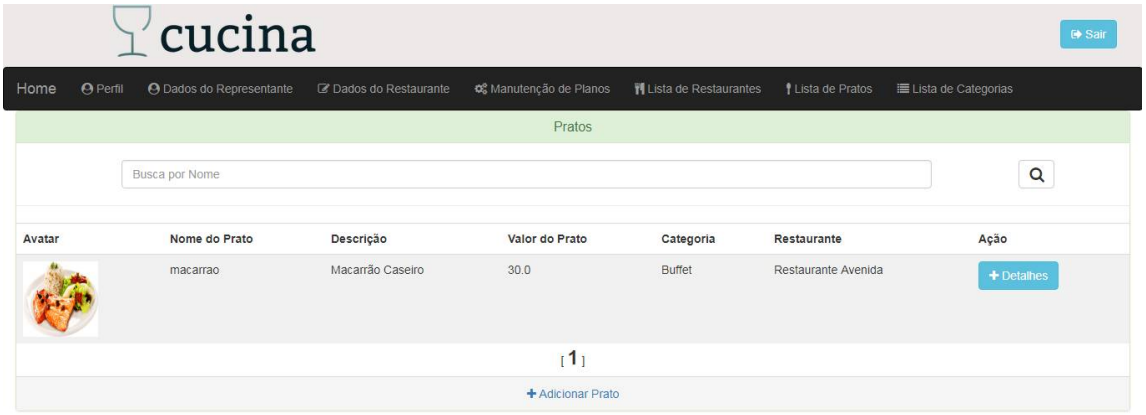

Figura 18 – Tela de listagem de pratos

Para visualizar informações que não estejam visíveis no *grid* com a listagem de pratos, o representante poderá clicar na opção "+ Detalhes" que exibe uma tela com informações específicas do prato selecionado, como, por exemplo, a imagem do prato, sua descrição detalhada, os ingredientes e o valor. Essa tela possui também o direcionamento para a tela de detalhes do restaurante por meio do *link* com o nome do restaurante que permite que o usuário obtenha informações de localização e do telefone do restaurante, por exemplo.

Lembrando que esta tela não permite ao representante de seu restaurante fazer alterações, isso é possível pela tela de perfil de restaurante que é disponível apenas para seu o representante pela opção de menu "Dados do Restaurante". A tela de detalhes do prato é disponibilizada para todos os perfis de usuários, desde que estejam cadastrados no sistema, usuários anônimos não tem permissão de acesso à essa tela, para isso devem se cadastrar.

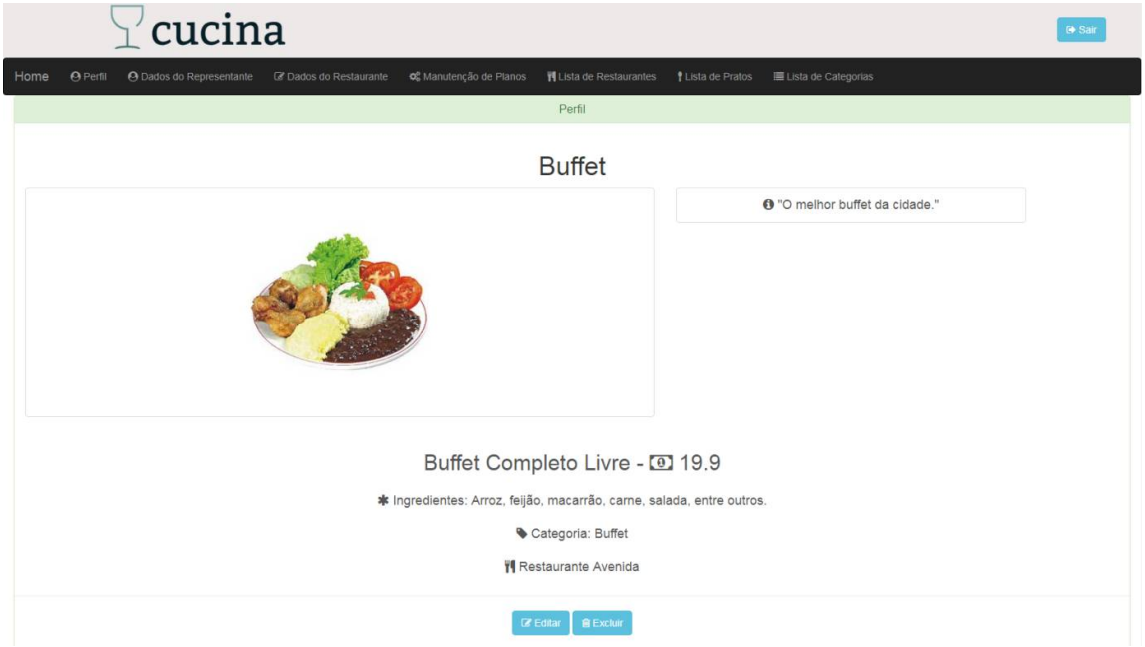

Figura 19 – Tela de detalhes do prato

A tela de lista de categorias, apresentada na Figura 20, é disponibilizada apenas para usuários com perfil de administrador e representante. No entanto, as opções de adicionar, editar e remover só serão disponibilizadas aos usuários de perfil administrador que são responsáveis pela manutenção de categorias. Os usuários de perfil de representante possuem acesso a essa lista apenas para identificar quais as categorias estão disponíveis para utilizarem em seus cadastros de pratos.

Figura 20 – Tela de listagem de categorias

O administrador tem a opção de excluir ou editar um plano na tela de lista de planos, conforme apresenta a Figura 21. Essa tela apresenta os dados do plano, como, tipo, descrição, valor, quantidade de pratos e qualificação para o restaurante. Além disso, possui os botões de excluir, editar e exibir mais detalhes do plano.

Figura 21 – Tela de manutenção de planos

A tela de perfil do plano, apresentada na Figura 22, é disponibilizada para os usuários administrador e representante. O administrador possui a opção de editar e excluir os dados do plano e o representante poderá contratar um novo plano ou migrar de plano, se possuir em seu restaurante uma quantidade igual ou menor de pratos cadastrados que o plano desejado lhe oferece.

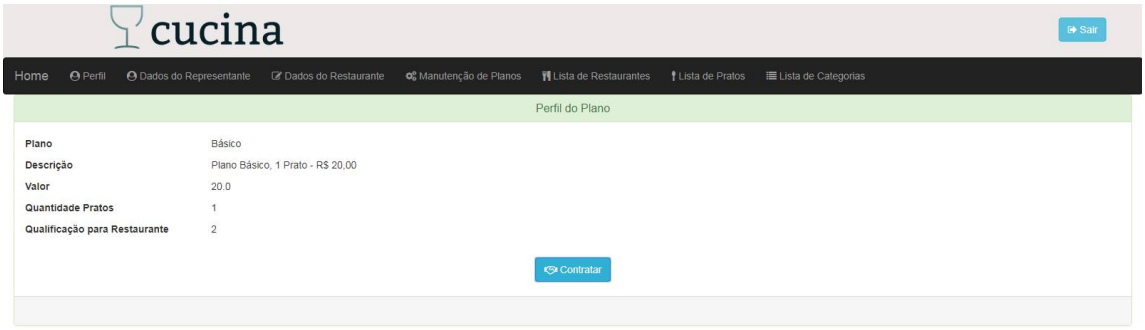

Figura 22 – Tela de perfil do plano

Ainda na tela de perfil de plano, caso o representante não possa contratar o plano, por ter uma quantidade de pratos cadastrada maior que a disponibilizada pelo plano, lhe será apresentada uma mensagem lhe informando sobre o ocorrido, apresentada na figura 23.

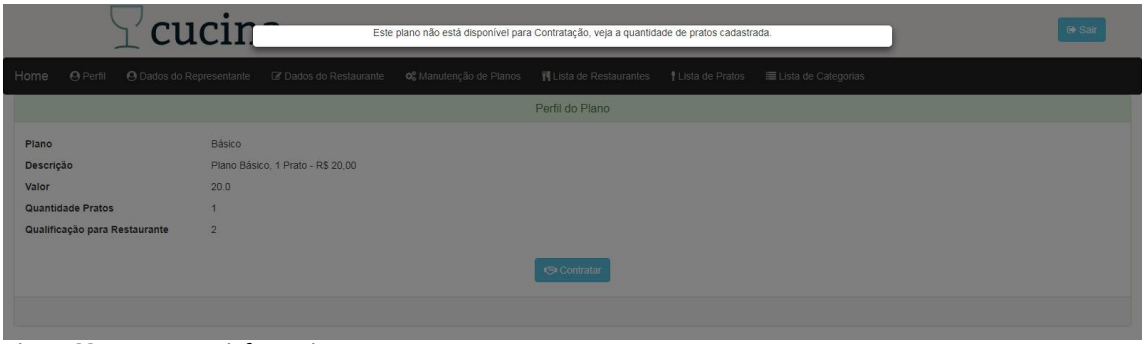

Figura 23 - Mensagem informativa

3.4 IMPLEMENTAÇÃO DO SISTEMA

Para a codificação do sistema foi realizada a instalação e a configuração do ambiente de desenvolvimento juntamente com os recursos necessários (*frameworks*, bibliotecas, *plugins*, etc.). Primeiramente foi realizada a instalação do Eclipse Neon e a instalação e configuração do servidor ApacheTomcat. Para iniciar o desenvolvimento do sistema foi criado um projeto Maven.

Como servidor de banco de dados foi instalado e configurado o MySQL e para criar e manipular o banco de dados (utilizando a linguagem SQL), como sistema de gerenciamento de banco de dados foi utilizado o MySql Workbench. Para a configuração do *framework* Spring Data, primeiramente foi necessário adicionar as dependências necessárias ao POM, através dessas dependências o maven fez download das devidas bibliotecas. Configurado o Spring Data como provedor de persistência.

Após a configuração Spring Data foram criadas as classes de modelo conforme o diagrama de entidade e relacionamento, juntamente com as devidas anotações necessárias utilizadas do JPA para realizar o mapeamento das classes, como apresentado na Listagem 2.

```
@Entity
 @Table(name = "restaurantes")
publicclassRestauranteextendsAbstractPersistable<Long> {

	@Column(nullable = false, length = 60)
private String nome;

	@CNPJ
	@Column(nullable = false)
private String cnpj;
```

```
	@Column(nullable = false, length = 200)
private String detalhes;

	@Column(nullable = false, columnDefinition = "LONGTEXT")
private String sobre;

	@Column(nullable = false)
private String endereco;

	@Column(nullable = false)
private String cidade;

	@Column(nullable = false)
private String estado;

	@Column(nullable = false)
private String pais;

	@Column(nullable = false)
private String telefone;

	@Column(nullable = false)
private Boolean status;

@Column(nullable = false)
private Integer qualificacao;

	@Column(name = "data_cadastro", nullable = false)
privateLocalDatedataCadastro;

	@Column(nullable = false, unique = true, length = 60)
private String permalink;

@OneToOne()
	@JoinColumn(name = "representante_id")
privateRepresentanterepresentante;

	@OneToMany(mappedBy = "restaurante")
privateList<Prato>pratos;

	@OneToOne(cascade = { CascadeType.PERSIST, CascadeType.REMOVE })
	@JoinColumn(name = "avatar_id")
privateAvataravatar;

	@Column
private Boolean ativo;

	@Column
private Integer qtdPratos;

	@OneToMany(mappedBy = "restaurante")
privateList<Comentario>comentarios;
```

```
	@Override
publicvoidsetId(Long id) {
super.setId(id);
	}
}
```

Listagem 2- Tabela Restaurante

A Listagem 3 apresenta a interface de repositório RestauranteRepository que estende a interface JpaRepository, que está sendo uma interface genérica do Spring Data que oferece suporte para a manipulação de dados, sem que o desenvolvedor necessite implementar vários métodos, alguns deles são: *save*, *delete*, *findAll*, entre outros.

```
package br.edu.utfpr.tcc.repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

publicinterfaceRestauranteRepositoryextendsJpaRepository<Restaurante, Long> {

	@Query("Select r from Restaurante r where r.nome like %?1% or r.cidade like %?1% and r.ativo = true")
	List<Restaurante>findByNomeLike(String nome);

Restaurantefindbyavatar(Avatar avatar);

Restaurantefindbypermalink(String permalink);

	@Query("Select r from Restaurante r where r.cidade like '%apanema%'")
	List<Restaurante>findByCidade();

Restaurantefindbyrepresentanteid(Long id);

	@Query("Select r from Restaurante r where r.ativo = true")
	List<Restaurante>findByAtivo();

List<Restaurante>findTop5ByOrderByQualificacaoDesc();

	List<Restaurante>findByNomeContainingIgnoreCaseOrderByNomeAsc(String nome);
}
```

Listagem 3- Classe RestauranteRepository

A Listagem 4 apresenta a classe de serviço RestauranteService, que é responsável pelas transações com o banco de dados. A anotação @Service faz com que se torne uma

classe de serviço do Spring Framework, que se encarrega do controle de transações no banco de dados. Ela também é responsável por fazer *rollback*, caso ocorra alguma falha durante a transação de dados. Esta classe também permite injeção de dependências, utilizada por meio da anotação @Autowired.

```
package br.edu.utfpr.tcc.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional(readOnly = true, propagation = Propagation.REQUIRED)
publicclassRestauranteService {

	@Autowired
privateRestauranteRepositoryrestauranteRepository;

publicList<Restaurante>findByAtivo() {
returnrestauranteRepository.findByAtivo();
	}

publicList<Restaurante>findByTop() {
returnrestauranteRepository.findTop5ByOrderByQualificacaoDesc();
	}

publicList<Restaurante>findAll() {
returnrestauranteRepository.findAll();
	}

publicList<Restaurante>findByCidade() {
returnrestauranteRepository.findByCidade();
	}

publicList<Restaurante>findByNomeLike(String nome) {
returnrestauranteRepository.findByNomeLike(nome);
	}

publicRestaurantefindByPermalink(String permalink) {
returnrestauranteRepository.findByPermalink(permalink);
	}

publicRestaurantefindById(Long id) {
returnrestauranteRepository.findOne(id);
	}

	@Transactional(readOnly = false)
publicvoiddelete(Long id) {
restauranteRepository.delete(id);
	}
```

```
	@Transactional(readOnly = false)
publicvoidsaveOrUpdate(Restaurante restaurante) {
if (restaurante.getId() == null) {
			System.out.println("caiu no save");
			restaurante.setQualificacao(0);
			restaurante.setAtivo(true);
			restaurante.setQtdPratos(0);
			save(restaurante);
		} else {
			System.out.println("caiu no update");
			update(restaurante);
		}
	}

	@Transactional(readOnly = false)
publicvoidatualizaQtdPratos(Restaurante restaurante) {
Restaurante persistente = restauranteRepository.findOne(restaurante.getId());
if (!persistente.getQtdPratos().equals(restaurante.getQtdPratos())) {
			persistente.setQtdPratos(restaurante.getQtdPratos());
		}
restauranteRepository.save(persistente);
	}
	@Transactional(readOnly = false)
publicvoidatualizaQualificacao(Restaurante restaurante) {
Restaurante persistente = restauranteRepository.findOne(restaurante.getId());
if (!persistente.getQualificacao().equals(restaurante.getQualificacao())) {
			persistente.setQualificacao(restaurante.getQualificacao());
		}
restauranteRepository.save(persistente);
	}

publicvoidupdate(Restaurante restaurante) {
Restaurante persistente = restauranteRepository.findOne(restaurante.getId());

if (!persistente.getNome().equals(restaurante.getNome())) {
persistente.setNome(restaurante.getNome());
		}
if (!persistente.getDetalhes().equals(restaurante.getDetalhes())) {
			persistente.setDetalhes(restaurante.getDetalhes());
		}
if (!persistente.getSobre().equals(restaurante.getSobre())) {
			persistente.setSobre(restaurante.getSobre());
		}
if (!persistente.getEndereco().equals(restaurante.getEndereco())) {
			persistente.setEndereco(restaurante.getEndereco());
		}
if (!persistente.getCidade().equals(restaurante.getCidade())) {
			persistente.setCidade(restaurante.getCidade());
		}
if (!persistente.getEstado().equals(restaurante.getEstado())) {
			persistente.setEstado(restaurante.getEstado());
		}
if (!persistente.getPais().equals(restaurante.getPais())) {
```

```
                persistente.setPais(restaurante.getPais());
            }
if (!persistente.getTelefone().equals(restaurante.getTelefone())) {
                persistente.setTelefone(restaurante.getTelefone());
            }
restauranteRepository.save(persistente);
    }

privatevoidsave(Restaurante restaurante) {
            restaurante.setQualificacao(0);
restaurante.setStatus(true);
            String permalink = ReplaceString.formatarPermalink(restaurante.getNome());
restaurante.setPermalink(permalink);
            restaurante.setDataCadastro(LocalDate.now());
restauranteRepository.save(restaurante);
    }

publicRestaurantefindByRepresentante(Long id) {
returnrestauranteRepository.findByRepresentanteId(id);
    }
publicRestaurantefindByAvatar(Avatar avatar) {
returnrestauranteRepository.findByAvatar(avatar);
    }
}
```

Listagem 4- Classe RestauranteService

A Listagem 5 apresenta a classe controladora RestauranteController, que é responsável pela comunicação das telas Java *Serve Page* (JSP) com as classes de *back-end*. Essa classe é controlada por meio da anotação @Controller que é uma anotação do Spring Framework. A anotação @RequestMapping é responsável por mapear a URL a qual o *controler*irá interagir. Esta classe também permite injeção de dependências, utilizada por meio da anotação @Autowired.

```
@Controller
@RequestMapping("restaurante")
publicclassRestauranteController {

    @Autowired
privateRestauranteServicerestauranteService;

    @Autowired
privateAvatarServiceavatarService;

    @Autowired
privateRepresentanteServicerepresentanteService;

    @Autowired
privatePlanoServiceplanoService;
```

```
	@Autowired
private PratoService pratoService;

	@Autowired
private CategoriaService categoriaService;

	@Autowired
private ComentarioService comentarioService;

	@InitBinder
public void init(WebDataBinder binder) {
		binder.registerCustomEditor(List.class, new PlanoEditorSupport(List.class, planoService));
	}

	@RequestMapping(value = "/search")
public ModelAndView search(@RequestParam(value = "nome") String nome) {
return new ModelAndView("restaurante/list", "restaurantes", restauranteService.findByNomeLike(nome));
	}

	@RequestMapping(value = "/list/{id}", method = RequestMethod.GET)
public ModelAndView listRestaurantesByRepresentante(@PathVariable("id") Long id, ModelMap map) {

		Long representanteId = representanteService.findById(id).getId();
map.addAttribute("restaurantes", restauranteService.findByRepresentante(representanteId));
map.addAttribute("representanteId", representanteId);
return new ModelAndView("restaurante/list", map);
	}

	@RequestMapping(value = "/perfil/{id}", method = RequestMethod.GET)
public ModelAndView perfil(@PathVariable("id") Long id) {
ModelAndView view = new ModelAndView();
Restaurante restaurante = restauranteService.findById(id);
List<Prato> pratos = pratoService.findByRestaurante(id);
		view.addObject("restaurante", restaurante);
		view.addObject("pratos", pratos);
view.setViewName("restaurante/perfil");
return view;
	}

	@RequestMapping(value = "/detalhe/save", method = RequestMethod.POST)
public String save(@ModelAttribute("prato") Prato prato) {
System.out.println("vai gravar");
pratoService.saveOrUpdate(prato);
return "redirect:/prato/perfil/" + prato.getId();
	}

	@RequestMapping(value = "/detalhe/{id}", method = RequestMethod.GET)
public ModelAndView detalhe(@PathVariable("id") Long id, @ModelAttribute("comentario") Comentario
comentario) {
ModelAndView view = new ModelAndView();
Restaurante restaurante = restauranteService.findById(id);
List<Prato> pratos = pratoService.findByRestaurante(id);
List<Comentario> comentarios = comentarioService.findByRestaurante(id);
```

```
            view.addObject("restaurante", restaurante);
            view.addObject("pratos", pratos);
            view.addObject("comentarios", comentarios);
            view.addObject("categorias", categoriaService.findAll());
            view.addObject("restaurantes", restauranteService.findAll());
            view.setViewName("restaurante/detalhe");
return view;
}

      @RequestMapping(value = "/update/{id}", method = RequestMethod.GET)
publicModelAndViewpreUpdate(@PathVariable("id") Long id, ModelMap map,
@AuthenticationPrincipal() UsuarioLogado logado) {
Restaurante restaurante = restauranteService.findById(id);

if (logado.getRepresentante().getRestaurante() == null) {
                  System.out.println("usuario nao e representante");
returnnewModelAndView("redirect:/restaurante/perfil/" + restaurante.getId());
} elseif (restaurante.getId() != logado.getRepresentante().getRestaurante().getId()) {
                  System.out.println("usuario e diferente do logado");
returnnewModelAndView("redirect:/restaurante/perfil/" + restaurante.getId());

}
            System.out.println("pode atualizar");
map.addAttribute("restaurante", restaurante);

returnnewModelAndView("restaurante/atualizar", map);
      }

      @RequestMapping(value = "/delete/{id}", method = RequestMethod.GET)
public String delete(@PathVariable("id") Long id, @AuthenticationPrincipal() UsuarioLogado logado) {
Restaurante restaurante = restauranteService.findById(id);
System.out.println(restaurante);
            System.out.println(restaurante.getId());

if (restaurante.getRepresentante().getId() != logado.getRepresentante().getId()) {
                  System.out.println("usuario diferente do logado");
return"redirect:/restaurante/perfil/" + restaurante.getId();
            } elseif (restaurante.getQtdPratos() != 0) {
                  System.out.println("Nao pode excluir");
return"redirect:/restaurante/perfil/" + restaurante.getId();
            }

Representante representante = representanteService.findById(logado.getRepresentante().getId());
            System.out.println("pode excluir");
            representante.setRestaurante(null);
representanteService.updateRestaurante(representante);

            System.out.println("Vai excluir restaurante do banco");
restauranteService.delete(id);

            System.out.println("Excluir restaurante");
            System.out.println("remover restaurante do representante");
```

```
logado.setRestaurante(null);
return"redirect:/";

	}

	@RequestMapping(value = "/list", method = RequestMethod.GET)
publicModelAndViewlistRestaurantes(ModelMap map) {
		map.addAttribute("restaurantes", restauranteService.findAll());
returnnewModelAndView("restaurante/list", map);
	}

	@RequestMapping(value = "/atualizar", method = RequestMethod.POST)
public String atualizar(@ModelAttribute("restaurante") Restaurante restaurante,
			@AuthenticationPrincipal() UsuarioLogado logado) {
restauranteService.saveOrUpdate(restaurante);
return"redirect:/restaurante/perfil/" + restaurante.getId();
}

	@RequestMapping(value = "/save", method = RequestMethod.POST)
public String save(@ModelAttribute("restaurante") @ValidatedRestaurante restaurante, BindingResult result,
			@AuthenticationPrincipal() UsuarioLogado logado,
			@RequestParam(value = "file", required = false) MultipartFile file) {

if (result.hasErrors()) {
return"restaurante/cadastro";
		}

Avatar avatar = avatarService.getAvatarByUpload(file);
if (logado.getId() != null) {
Representante representante = representanteService.findById(logado.getRepresentante().getId());
			restaurante.setRepresentante(representante);
			restaurante.setAvatar(avatar);
restauranteService.saveOrUpdate(restaurante);
			representante.setRestaurante(restaurante);
representanteService.updateRestaurante(representante);
			logado.setRestaurante(restaurante);
}

return"redirect:/restaurante/perfil/" + restaurante.getId();
	}

	@RequestMapping(value = "/add", method = RequestMethod.GET)
publicModelAndViewcadastro(@ModelAttribute("restaurante") Restaurante restaurante,
			@AuthenticationPrincipal() UsuarioLogado logado) {

Representante representante = representanteService.findByUsuario(logado.getId());

if (representante == null) {
returnnewModelAndView("redirect:/representante/add");
}

		restaurante = restauranteService.findByRepresentante(logado.getRepresentante().getId());
if (restaurante == null) {
```

```
                System.out.println("foi para a tela de cadastro");
returnnewModelAndView("restaurante/cadastro");
            }
returnnewModelAndView("redirect:/restaurante/perfil/" + restaurante.getId());
}
}
```

Listagem 5- Classe RestauranteController

O código da Listagem 6 mostra como uso do Spring Framework agilizou o desenvolvimento do sistema, fornecendo uma forma de implementar os controladores de maneira ágil e prática por meio do uso de anotações facilitando especialmente a codificação de acesso aos métodos por requisições em *Uniform Resource Locator* (URLs) específicas.

A codificação das interfaces de visualização do cliente foi realizada utilizando as tecnologias HTML5, CSS e JQuery em arquivos JSP. Para manipular informações em um arquivo JSP foi utilizado uma biblioteca de *tags* denominada *JavaServer Pages Standard Tag Library* (JSTL) que possibilita manipular arquivos JSP que são renderizados em HTML e enviados para visualização pelo cliente.

A Listagem 6 apresenta o uso do JSTL para renderizar uma lista dos restaurantes cadastrados. Este trecho não apresenta o cabeçalho, menu e rodapé da página, apenas o conteúdo de listagem dos restaurantes.

```
<divclass="container-fluid"align="center">
<divclass="row">
<divclass="col-md-12">
<divclass="panelpanel-success">
<divclass="panel-heading">
<h1class="prop">Lista de Restaurantes</h1>
</div>
<br>
<c:forEachvar="restaurante"items="${restaurantes}"varStatus="i">
<dl>
<dt>
<divclass="rowform-group">
<divclass="col-md-12col-xs-12">
<a
href="<c:url value="/avatar/load/${restaurante.avatar.id}" />"
                                title="Clique aqui para ampliar"><img
class="col-md-6col-xs-12thumbnail"
src="<c:url value="/avatar/load/${restaurante.avatar.id}" />" /></a>
```

```
<h3>
<iclass="fafa-cutlery"aria-hidden="true"></i>
${restaurante.nome }

</h3>
<h5>
<iclass="fafa-info-circle"aria-hidden="true"></i>
                                "${restaurante.sobre}"
</h5>
<h5>
<iclass="fafa-map-marker"aria-hidden="true"></i>
${restaurante.endereco },${restaurante.cidade },
                                ${restaurante.estado }, ${restaurante.pais }
</h5>
<h5>
<iclass="fafa-phone"aria-hidden="true"></i>
${restaurante.telefone }
</h5>
<h5>
<iclass="fafa-trophy"aria-hidden="true"></i>
${restaurante.qualificacao }
</h5>
<c:urlvar="perfil"
value="/restaurante/detalhe/${restaurante.id }"/>
<ahref="${perfil }"title="+ Detalhes">
<buttontype="button"class="btnbtn-info">
<iclass="fafa-plus"aria-hidden="true"></i> Detalhes
</button>
</a>

</div>
</div>
</dt>
<hr>
</dl>
</c:forEach>
<divclass="panel-footer">
<ahref="<c:url value="/"/>"><iclass="fafa-home"
aria-hidden="true"></i> Página Inicial
</a><ahref="<c:url value="/"/>"><iclass="fafa-refresh"
aria-hidden="true"></i> Atualizar Página
</a>
</div>
</div>
</div>
</div>
</div>
```

Listagem 6- Lista de Restaurantes

A Listagem 7 apresenta o trecho de estruturação da tela do perfil de plano que apresenta as informações do plano validando o tipo de usuário autenticado. Se o usuário for

um representante será apresentado o botão “Contratar” (caso, este enquadre nas validações de quantidade de pratos cadastrados), senão não será apresentado nenhum botão para este tipo de usuário. Caso o usuário autenticado seja um administrador, será apresentado o botão de alterar e excluir plano.

```
<divclass="container-fluid"align="center">
<divclass="row">
<divclass="col-md-12">
<divclass="panelpanel-success">
<divclass="panel-heading">
<h1class="panel-titlepanel-success">Perfil do Plano</h1>
</div>
<br>

<divclass="rowform-group">
<labelclass="col-md-2col-xs-12control-label">Plano</label>
<divclass="colcol-md-10col-xs-12">
<outputclass="form-control">${plano.plano }</output>
</div>
</div>

<divclass="rowform-group">
<labelclass="col-md-2col-xs-12control-label">Descrição</label>
<divclass="colcol-md-10col-xs-12">
<outputclass="form-control">${plano.descricao }</output>
</div>
</div>

<divclass="rowform-group">
<labelclass="col-md-2col-xs-12control-label">Valor</label>
<divclass="colcol-md-10col-xs-12">
<outputclass="form-control">${plano.valor }</output>
</div>
</div>

<divclass="rowform-group">
<labelclass="col-md-2col-xs-12control-label">Quantidade
Pratos</label>
<divclass="colcol-md-10col-xs-12">
<outputclass="form-control">${plano.qtdPratos }</output>
</div>
</div>

<divclass="rowform-group">
<labelclass="col-md-2col-xs-12control-label">Qualificação
                    para Restaurante</label>
<divclass="colcol-md-10col-xs-12">
<outputclass="form-control">${plano.qualificacao }</output>
</div>
</div>
<br>
```

```
<div>
<security:authorizeaccess="hasAuthority('ADMIN')">
<c:urlvar="editar"value="/plano/update/${plano.id }"/>
<ahref="${editar }"title="Editar">
<buttontype="button"class="btnbtn-info">
<iclass="fafa-pencil-square-o"aria-hidden="true"></i> Editar
</button>
</a>

<c:urlvar="excluir"value="/plano/delete/${plano.id }"/>
<ahref="${excluir }"title="Excluir">
<buttontype="button"class="btnbtn-info">
<iclass="fafa-trash-o"aria-hidden="true"></i> Excluir
</button>
</a>
</security:authorize>

<security:authorizeaccess="hasAuthority('REPRESENTANTE')">
<c:if
test="${logado.getRepresentante().getRestaurante().getQtdPratos() < plano.qtdPratos
|| logado.getRepresentante().getRestaurante().getQtdPratos() == plano.qtdPratos}">
<c:urlvar="save"value="/plano/contratar"/>
<form:formmodelAttribute="plano"action="${save }"
method="post">
<inputtype="hidden"value="${plano.id}"name="id">

<divclass="">
<buttonclass="btnbtn-info"type="submit">
<iclass="fafa-handshake-o"aria-hidden="true"></i>
                                    Contratar
</button>
</div>
</form:form>
</c:if>
</security:authorize>

</div>
<br>
<divclass="panel-footer">
<ahref="<c:url value="/"/>"><iclass="fafa-home"aria-hidden="true"></i> Página Inicial
</a>
</div>
</div>
</div>
</div>
</div>
```

Listagem 7- Perfil de Plano

4 CONSIDERAÇÕES FINAIS

Este trabalho apresentou o desenvolvimento de um sistema *web* de divulgação e representação de restaurantes, oferecendo maior *marketing* e facilidade de divulgação de propaganda, centralizando a busca do cliente em apenas um site.

O *framework* Spring Frameworkse mostrou uma ferramenta importante para o desenvolvimento ágil de um sistema *web* em Java. O auxílio de *frameworks* e bibliotecas como o Bootstrap e JQueryfoi fundamental para a produtividade no desenvolvimento de interfaces intuitivas, dinâmicas e responsivas, agilizando, assim, o desenvolvimento do trabalho.

O projeto não foi publicado por ainda necessitar de um investimento, em fundos e desenvolvimento, pois ainda necessita de uma integração para uma forma de pagamento eletrônico para comportar os possíveis contratos de representantes que buscarem essa solução.

REFERÊNCIAS

GOMES, Helton S. **Internet chega pela 1ª vez a mais de 50% das casas no Brasil, mostra IBGE.** 2016. Disponível em: <http://g1.globo.com/tecnologia/noticia/2016/04/internet-chega-pela-1-vez-mais-de-50-das-casas-no-brasil-mostra-ibge.html> Acesso em: 23 nov. 2016.

FOLHA DE SÃO PAULO. **Brasil é o terceiro pais do mundo que fica mais tempo on-line no celular**. 2015. Disponível em: <http://www1.folha.uol.com.br/tec/2015/09/1679423-brasil-e-terceiro-pais-do-mundo-que-fica-mais-tempo-on-line-no-celular.shtml>. Acesso em: 23 nov. 2016.

BEZERRA, Carlos Adriano Tanaka; LIMA Mateus Machado de; VIEIRA, Osmar André. Cardápio Digital, Tecnologia aliada à Gastronomia. Editora. Revolução eBook, 2015.

HOUAISS, Antonio; VILLAR, Mauro de Salles. **Dicionário Houaiss da Língua Portuguesa**. Rio de Janeiro, Objetiva, 2004.